PARTITION & DECODE: AN IMPLICIT INTERNAL REPRESENTATION FRAMEWORK

by

Kaleab Alemayehu Kinfu

A thesis submitted to Johns Hopkins University

in conformity with the requirements for the degree of

Master of Science in Engineering

Baltimore, Maryland

August, 2021

Abstract

In machine learning, we study and build algorithms designed to leverage data to improve performance on a set of tasks. Some approaches, such as modern deep networks, explicitly learn internal representations of the data. Other approaches, such as support vector machines, do not explicitly learn such internal representations. We introduce "Partition and Decode" (P&D), a framework that formalizes an implicit internal representation of a large number of supervised and unsupervised machine learning methods, including decision trees and forests (including gradient boosting trees), and deep networks. The key realization is that each of these approaches explicitly or implicitly partitions feature space into parts, and therefore, the training data can be encoded by which part it resides in. Moreover, we empirically demonstrate that our new complexity metric based on some matrix norm of the internal representation correlates more strongly with the generalization gap than previously proposed measures, while also being able to be applied to non-deep learning-based approaches.

Primary Reader and Advisor: Professor Joshua T. Vogelstein

Secondary Readers: Professor Carey E. Priebe and Professor Soledad Villar

Acknowledgments

First and foremost, I would like to praise the heavenly father, Almighty God, whose many blessings have made me who I am today; and St. Virgin Mary, who have always interceded on my behalf in my prayers to her son Jesus Christ.

This thesis would not have been possible without the guidance and help of many people who in one way or another contributed. First of all, I would like to express my deepest gratitude to my advisor Professor Joshua Vogelstein for sparking my interest in this research and giving me the opportunity and guidance to pursue my research within such an outstanding environment as the NeuroData lab with very talented, open-minded and highly motivated people. One could not hope for a better advisor and working environment. Moreover, I would like to extend my sincere gratitude and credit to Professor Joshua Vogelstein, Professor Carey Priebe, Professor Florian Engert, and Dr. Christopher White for their initial ideas and work on this topic as well as to Will LeVine, Ronan Perry, Ningyuan (Teresa) Huang, Jayanta Dey, Michael Ainsworth, and Weiwei Yang for their great contribution to the research work.

I am also very grateful to Professor Carey Priebe and Professor Soledad

Villar for serving in my thesis committee. Besides, I want to thank my classmates and all teaching and administrative staff at Johns Hopkins who have been so supportive and cooperative over the course of the program. Finally yet importantly, I would like to extend my appreciation to my family for all their unconditioned love and support.

Dedication

Dedicated to the loving memory of my dear uncle, Tafere Kinfu Kassaye.

Table of Contents

Abstract ii

Acknowledgments iii

Dedication v

Table of Contents vi

List of Tables viii

List of Figures ix

1 Introduction 1

2 Background and Related Works 4

2.1 Classification 4

2.2 Decision Forests 6

2.3 Deep Networks 8

3 Partition & Decode Framework 12

3.1 Learning by partition and decode 14
3.1.1 Experimental setup 14
3.1.2 Learning partitions 14
3.2 Internal representations of DN and DF 16
3.2.1 Quantifying complexity in ReLU deep nets 19
3.2.2 An empirical investigation of the P&D bounds 21
3.2.3 Generalization in Deep Networks 25
3.2.3.1 Evaluation criteria 28

4 Discussion and Conclusion **31**

Bibliography **36**

List of Tables

3.1 Complexity measures, reference to the equation, and the rank-correlation coefficients with DN models (increasing depth) trained on the Gaussian XOR dataset. 29

3.2 Complexity measures, reference to the equation, and the rank-correlation coefficients with DN models (both increasing depth and increasing width) trained on the Gaussian XOR dataset. . 30

List of Figures

2.1 A tree in the forest 7

2.2 A shallow deep network 10

3.1 Gaussian XOR samples used as a benchmark 15

3.2 Visualizations of partitions 16

3.3 Visualizations of the Polytope Compositions for DF/DN . . . 17

3.4 Performance of DN with two different learning settings 23

3.5 Effect of depth vs width on the generalizability of a DN models with the same number of parameters 25

Chapter 1

Introduction

Over the last six decades, computing systems have become an indispensable part of our lives. We use them in virtually all sectors to boost productivity by thousands of times, thus saving huge amounts of human working hours. They are highly utilized in numerous fields such as health, enhancing our day-to-day activities and life quality. As the technology advances, people depend increasingly on computers and algorithms in daily routine. This attainment is mainly because of the ability of modern computers to process excessively large amounts of data, availability of efficient algorithms, and the growing capability of computing systems.

As a result of these significant advances, machine learning systems are able to perform complex tasks by learning directly from examples, data, and experience, rather than following pre-programmed step-by-step rules. Therefore, recent years have witnessed prominence of machine intelligence, and the promises to save lives, address global challenges, and boost the global economy through increasing productivity; while doing so it also enhances our health, productivity, and well-being.

Deep networks (DNs) and decision forests (DFs) are the two dominant state-of-the-art learning methods in modern machine learning, each with its own theoretical and computational benefits. Deep networks, on the one hand, empirically outperform all other methods on *structured* data problems with large data corpora, such as images (Krizhevsky, Sutskever, and Hinton, 2012), text (Brown et al., 2020), speech (Zhang et al., 2020), and autonomous control (Statt, 2019). A structured data sample is any sample in which the relative position of features matters, so that permuting the feature indices discards some of the information. For example, to encode a 1-dimensional image one requires the relative two-dimensional position of each pixel (which are the feature indices), as well as the magnitudes. Permuting the feature indices discards information about the image, in fact, it discards 2/3's of the content.

Ensemble methods, on the other hand, empirically outperform all other methods on unstructured or *tabular* data problems, which are common in biomedical applications for example. More specifically, random forests and gradient boosting trees have dominated all other methods in papers comparing various methods on real datasets (Caruana and Niculescu-Mizil, 2006; Caruana, Karampatziakis, and Yessenalina, 2008; Fernández-Delgado et al., 2014), as well as real-world competitions (Chen and Guestrin, 2016).

The fact that these two approaches dominate in complementary settings have motivated a number of efforts to combine the best of both worlds. For

example, (Patel, Nguyen, and Baraniuk, 2015) pointed out that under certain assumptions, both DN and DF can be cast as max-sum message passing networks. More recently, others combined deep networks with random forests (Zhou and Feng, 2018), or made random forests differentiable (Shen et al., 2019), or used other techniques to make random forests end-to-end trainable (Carreira-Perpinan and Tavallali, 2018; Hehn and Hamprecht, 2019). Moreover, Priebe et al., 2020 studied the relationship between the internal representations that the two approaches learn. We extend the work by introducing "Partition and Decode" (P&D), a framework that formalizes an implicit internal representation of such learning methods. The key realization is that these approaches explicitly or implicitly partitions feature space into parts, and therefore, the training data can be encoded by which part it resides in. In this thesis, we illustrate the commonalities of their representations, and leverage such commonalities to explain certain phenomena, such as the double descent paradox. We close by demonstrating a complexity measure based on matrix norm of the internal representation strongly correlates with generalization than existing state-of-the-art complexity measures.

Chapter 2

Background and Related Works

2.1 Classification

Classification involves taking set of observations or input data and assigning it with corresponding class labels. The task is to design a classification procedure by training a model based on annotated data that will serve as a means for estimating the class of new observations. The classical statistical formulation of the classification problem consists of

$$(X, Y), (X_1, Y_1), \cdots, (X_n, Y_n) \overset{iid}{\sim} \mathcal{F}_{XY},$$

where $\mathcal{T}_n = \{(X_i, Y_i)\}_{i=1}^n$ is the training data and (X, Y) represents the to-be-classified test observation X with unobserved class label Y. We consider the simplest setting in which a feature $X \subseteq \Re^d$ (the space of feature vectors) and Y is a class label in $\{0, 1\}$. The goal is to learn a classification rule $g_n = g(\cdot; \mathcal{T}_n)$ that maps the feature vectors to class labels such that the probability of incorrect classification $L(g_n) = P[g(X; \mathcal{T}_n) \neq Y | \mathcal{T}_n]$ is small. An optimal

classifier g^* is the Bayes classifier that can be defined as:

$$g^*(X) = \underset{y \in \{0,1\}}{\operatorname{argmax}} P(Y = y|X)$$

and $L^* = L(g^*)$ is the Bayes optimal probability of misclassification.

Stone's Theorem for universally consistent classification (Stone, 1977; Devroye, Györfi, and Lugosi, 1997) demonstrates, loosely speaking, that a successful classifier can be constructed by first partitioning the input space into cells – the partition depends on n – such that the number of training data points in each cell goes to infinity but slowly in n, and then estimating the posterior $\eta(x) = P[Y = 1|X = x]$ locally by voting based on the training class labels associated with the training feature vectors in cell $C(x) \subset \Re^d$ in which the test observation falls. That is, letting $S(x) = \{X_1, \cdots, X_n\} \cap C(x)$ be the subset of X_i's falling in cell $C(x)$, and letting $N(x) = |S(x)| = \sum_i I\{X_i \in C(x)\}$ be the cardinality of this set, we consider the posterior estimate $\widehat{\eta}(x) = (1/N(x)) \sum_{i:X_i \in C(x)} I\{Y_i = 1\}$. Then, under some technical conditions on the manner of choosing the sequence of partitions $\mathcal{P}_n = \{C_{n,1}, \cdots, C_{n,K_n}\}$, the plug-in rule $g(X; \mathcal{T}_n) = I\{\widehat{\eta}(X) > 1/2\}$ is universally consistent: $L(g_n) \to L^*$ almost surely for any $\mathcal{F}_{XY}$.

In the context of our formal definition of the classification problem, we provide a unified description of the two dominant methods, i.e. the decision forests (DF) and the deep networks (DN).

2.2 Decision Forests

Decision forests, including random forests and gradient boosting trees, are ensemble of decision trees that demonstrate state-of-the-art performance in a variety of machine learning settings. Forests have typically been implemented as ensembles of axis-aligned decision trees – trees that split along canonical feature dimensions only – but modern extensions employ axis-oblique splits (Breiman, 2001; Criminisi and Shotton, 2013; Athey, Tibshirani, and Wager, 2019; Tomita et al., 2020). Learning a decision tree from training data is essentially learning a hierarchical, tree-structured partitioning of the input space and then learning to estimate the label within each leaf node. During inference, we simply traverse down the decision tree from the root to the leaf node and classify.

A more formal definition of decision forests is: given the training data $\mathcal{T}_n$, each tree t in a forest constructs a partition $\mathcal{P}_{n,t}$ by successively splitting the input space based on a subset of the data and then choosing a hyperplane split at each node based on a subset of the data using some split criterion that maximizes a measure of information gain. A classical approach is minimizing impurity, such as the Gini impurity score: a number between 0 and 0.5 that indicates the likelihood of incorrect classification. A leaf node is created when the partition reaches a stopping criterion, i.e. no features result in any information gain, the impurity score is below a threshold or has less than a minimum number of observations (Hastie, Tibshirani, and Friedman, 2001). The details provide ample fodder for myriad implementations. However, it is not straight-forward to depict the visualization of input space $\Re^d$ for $d > 2$,

but Figure 2.1 illustrates the idea.

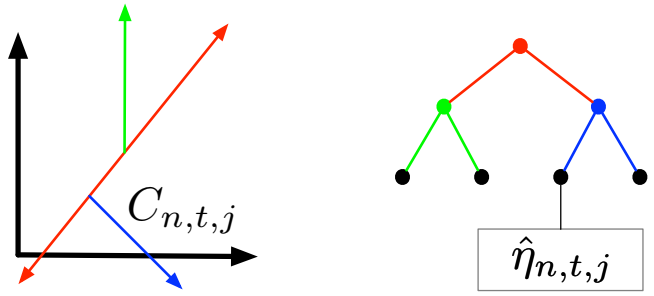

Figure 2.1: A tree in the forest. Given the random subset $\mathcal{T}_{n,t}$ of the training data allocated to tree t, the root node (red) performs a hyperplane split of $\Re^d$ based on a random subset of dimensions; the two daughter nodes (green and blue) split their respective partition cells based on a separate random subset of dimensions allocated to each node; etc. In the end, this tree results in a partition of $\Re^d$ with the leaf nodes corresponding to partition cells for which training data class labels yield local posterior estimates. The forest classifies the test observation feature vector X by voting over trees using the cells $C_{n,t}(X)$ in which X falls. Adapted from (Priebe et al., 2020).

The depth of each tree is the length of the unique path from the root to the deepest leaf. It is a function of n and involves a tradeoff between leaf purity and regularization to avoid overfitting. Hence, each tree of the forest results in a partition $\mathcal{P}_{n,t}$ of $\Re^d$ and each partition cell, i.e. leaf nodes in every tree, admits a posterior estimate $\widehat{\eta}_{n,t,j} = (1/N_{n,t,j}) \sum_{i:X_i \in C_{n,t,j}} I\{Y_i = 1\}$ based on the class labels of the training data feature vectors that fall into cell $C_{n,t,j}$.

The conceit of the forest is that of resampling: by choosing a separate subset $\mathcal{T}_{n,t}$ of the training data for each tree t, the ensemble forest posterior estimate is superior to that of any individual tree. Regardless, the overall classifier is seen to be a partition and vote scheme, and under appropriate conditions on

the manner of choosing the partitions $\mathcal{P}_{n,t}$ we have $L(g_n^{DF}) \to L^*$.

2.3 Deep Networks

Deep networks (DN) are extraordinarily popular and successful in modern machine learning (LeCun, Bengio, and Hinton, 2015; Sze et al., 2017; Montúfar et al., 2014; Montúfar, 2017). It became the leading technology behind numerous quantum leaps in many fields. The general and robust ability of learning representations from raw data is the main reason why it has opened the door to a broader range of applications, whose complexity of the phenomenon was even unimaginable to solve a few decades ago.

The term "Deep" corresponds to the deep multi-layer architecture of feed-forward network learning deep hierarchies of learned features, which is crucial, specifically for high-dimensional raw data. Every layer constitutes an elementary affine transformation of the output of the previous layer and a non-linear activation function. In theory, such architecture is expected to allow the model to approximate any function of interest to some degree of accuracy (Hornik, Stinchcombe, and White, 1989). The output layer of the structure represents the space of desired outputs, which depends on the function the network estimates. For instance, binary classification can be represented by a singular value and multi-class classification by a vector of probabilities.

As already mentioned, a sequence of linear layers (affine mappings), each followed by a non-linear activation function forms the standard DN model. More formally, we can denote the input vector by X, hidden linear layers by

ℓ_i, activation functions by σ_i, and the output layer by o, thus, a d-layer feed-forward network can be expressed as a function composition in the following way:

$$\tilde{X}_d = \ell_0 \circ \sigma_0 \circ \ell_1 \circ \sigma_1 \circ \cdots \ell_{d-1} \circ \sigma_{d-1} \circ o(X),$$

where $\circ$ denotes functional composition and $\tilde{X}_d$ is the result obtained by the d-layer feed-forward network.

Linear layers in these notations are simply the following matrix operation: $\ell_i = (\tilde{X}_{i-1}^T W_{i-1} + b_i)$, where the matrix W is called neural weights, and b is the vector of biases. Weights and biases form the parameter space of the network. As depicted in Figure 2.2, each internal node $v_{\ell,k}$ in layer ℓ of the network gathers inputs $\tilde{x}_{\ell-1,j}, j = 1, \cdots, n_{\ell-1}$ from the previous layer, weighted by $w_{\ell-1,j,k}, j = 1, \cdots, n_{\ell-1}$, and outputs

$$\tilde{x}_{\ell,k} = \sigma_{\ell,k}(\sum_j \tilde{x}_{\ell-1,j} w_{\ell-1,j,k}) = \sigma_{\ell,k}(\tilde{X}_{\ell-1}^T W_{\ell-1,k}) = \sigma(\tilde{X}_{\ell-1}^T W_{\ell-1,k} + b_{\ell,k}).$$

The activation functions, principally, can be almost any everywhere differentiable function, which introduces non-linearity to the output of the node, defined over the vector space. However, Rectified Linear Unit (ReLU), i.e. $\max(0, \cdot) = (\cdot)^+$, is mostly used for reasons like numerical stability and empirically proven effectiveness. It also provides the best performance over a wide range of tasks, as it does not saturate and the gradient is always high (equal to 1) if the node activates. As long as it is not a dead node, successive updates are also fairly effective.

When using ReLU as the activation function σ, each node performs a

hyperplane split based on a linear combination of its inputs, passing a non-zero value forward if the input is on the preferred side of the hyperplane; data in the cell defined by the collection of polytopes induced by nodes $\{v_{\ell-1,j}\}$, weighted, falls into node $v_{\ell,k}$ and is output based on a partition refinement. Letting $\tilde{X}_\ell = [\tilde{x}_{\ell,1}, \cdots, \tilde{x}_{\ell,n_\ell}]^T$ be the output of layer ℓ and $W_{\ell,j} = [w_{\ell,1,j}, \cdots, w_{\ell,n_\ell,j}]^T$ be the weights from layer ℓ to the jth node in layer $\ell+1$, node $v_{\ell+1,j}$ sees $\tilde{X}_\ell^T W_{\ell,j}$ and outputs $\tilde{x}_{\ell+1,j} = (\tilde{X}_\ell^T W_{\ell,j} + b_{\ell+1,j})^+$. Thus a node in the last internal layer corresponds to a union of hyperplane-induced partition cells, defined via composition of all the nodes earlier in the network.

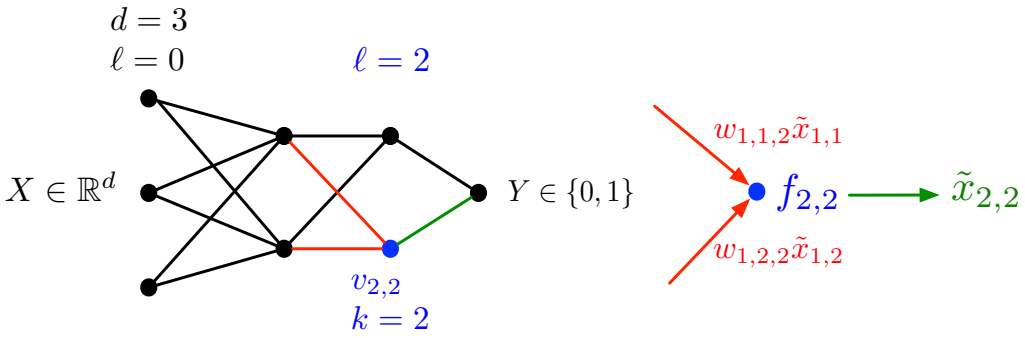

Figure 2.2: A (shallow) deep network. Given the training data $\mathcal{T}_n$, the $X_i \in \Re^d$ are passed through the network. At layer ℓ and node $v_{\ell,k}$ (the blue node is $v_{2,2}$) the inputs $w_{\ell-1,j,k}\tilde{x}_{\ell-1,j}$ (red) are transformed via hyperplane activation function $\sigma_{\ell,k}$ and output as $\tilde{x}_{\ell,k}$ (green). Thus node $v_{2,2}$ receives non-zero input $w_{1,j,2}\tilde{x}_{1,j} = w_{1,j,2}(X^T W_{0,j} + b_{1,j})^+$ from node $v_{1,j}$ if and only if the linear combination of the multivariate X, $X^T W_{0,j}$, is on the preferred side of hyperplane defined by $\sigma_{1,j}$ (and weight $w_{1,j,2}$ is non-zero). The output of node $v_{2,2}$ is $\tilde{x}_{2,2} = \left(\begin{bmatrix} (X^T W_{0,1} + b_{1,1})^+ \\ (X^T W_{0,2} + b_{1,2})^+ \end{bmatrix}^T W_{1,2} + b_{2,2} \right)^+$; that is, $v_{2,2}$ provides a further hyperplane refinement of $\Re^d$. Adapted from (Priebe et al., 2020).

For conceptual unification with DF, consider passing all the training data through the network; the collection of X_i's falling into each of the nodes in

the penultimate layer, together with their class labels, induces local posterior estimates for each of these cells. As with a single tree in a DF wherein the test observation feature vector X, suitably transformed, falls into one and only one leaf partition cell, (Montúfar et al., 2014) argues that DN with ReLu activation functions fold space such that a subset of $\Re^d$ maps to exactly one fold. Thus for a DN the input X falls into a final network partition cell in the last internal layer. Unlike in DF, for a DN the input X activating a node in the penultimate layer does not uniquely specify which partition cell X falls into; rather, it merely indicates that X falls into the set of partition cells corresponding to that node. For this reason, whereas in a DF each X activates only a single leaf node, in a DN each X can activate many (even all) cells in the penultimate layer, though with different activation energies. Thus while for a DF the ensemble is realized by voting over a forest of trees, for a DN we have membership in a single final network partition cell. On the other hand, each cell in a DF is constructed via a simple local splitting process, while for a DN a complex parameter estimation solution is employed to tailor the individual final network partition cells to the training data. In other words, both DF and DN can be seen to use the same representation space, though they achieve their particular representation via different estimation ("learning") algorithms.

The parameters for this partition and decode scheme are estimated ("learned") so as to make the network input/output fit the training data $\mathcal{T}_n$, with much current effort involving aspects of regularization to alleviate overfitting and capacity as a function of the number of layers and nodes per layer.

Chapter 3

Partition & Decode Framework

In machine learning methods, model complexity is an essential basic problem. Model complexity indicates how well a model learns particular problems and data, as well as the learned model's generalization ability on unobserved data. The complexity of a learned model can be affected by the model architecture, data distribution, data complexity, and information volume. Model complexity measures are useful for interpretation and analysis of internal mechanisms of models.

Parameter counting is often used as a proxy for model complexity (Zhang et al., 2017; Belkin et al., 2019). However, counting the algorithmic parameters is not a particularly sensible thing to do for measuring algorithmic complexity in many modern machine learning methods (including deep learning), and sometimes cannot even be done. For example, 1-nearest neighbors and k-nearest neighbors literally have zero parameters. In contrast, support vector machines and Gaussian process can have an infinite number of parameters. Decision trees are often 'parameterized' by the number of splits they make. One could count the number of parameters in a random forest by adding

up the total number of splits. But, what if two trees make the same split, do we count that parameter twice? Similar, in deep networks the number of parameters are often counted. But, if a node (in the computation graph) has a bias that renders it impossible to activate for any data, do we count all its weights as parameters as well?

We are interested in a particularly popular and important subset of the algorithmic modeling approaches which we deem 'Partition and Decode' (P&D). The key to partition and decode approaches is that they partition features space into a finite set of parts that jointly span the sample space, and then apply a (typically simple) function independently on each part to make a prediction. P&D is a quite large and general class of functions, including 1-nearest neighbors, k-nearest neighbors, kernel machines using kernels with finite support, decision trees, decision forests, and deep networks with ReLU activation functions. In contrast, algorithms that are not P&D include polynomial regression, kernel machines using kernels with unbounded support, and deep networks with sigmoidal activation functions. Moreover, P&D approaches are not limited to supervised learning: histograms and variational auto-encoders with ReLU activation functions are P&D approaches as well, whereas kernel density estimation using kernels with unbounded support are not. Intriguingly, some P&D approaches are interpolators (including 1-NN and decision forests) and others are not necessarily (including k-NN and some deep nets depending on the data and learning algorithm) (Belkin, Hsu, and Mitra, 2018). In this chapter, we discus how P&D methods learn by partitioning feature space and decode the internal representation. Moreover, we propose

an approach to quantifying the complexity of P&D approaches that does not explicitly depend on the parameters.

3.1 Learning by partition and decode

3.1.1 Experimental setup

To provide a concrete illustrative numerical example, consider the following experimental setup. We generate a two-dimensional Gaussian XOR dataset (see Figure 3.1) with four spherically symmetric Gaussians. Class 0 has two Gaussians with centers $(-1, -1)$ and $(1, 1)$, whereas class 1 has two Gaussians with centers $(1, -1)$ and $(-1, 1)$. All Gaussians have an equal likelihood of being sampled. In other words, $y \overset{iid}{\sim} \mathbf{B}(\pi)$ $x \overset{iid}{\sim} \sum_{i=1}^{4} \pi_{y,i} \mathcal{N}(\mu_i, \mathbb{I})$, where $\mathbb{I}$ denotes the identity matrix, and the four μ_i's are as described above.

We use the two-dimensional Gaussian XOR example with 4096 training samples and 1000 testing samples to illustrate this point. Note that because the number of dimensions for this example is two, and there are no noise dimensions, these results can not be explained away with the benign overfitting (Bartlett et al., 2020).

3.1.2 Learning partitions

Figure 3.2 shows the partitions learned by the two methods: DF and DN. More formally, let N_l be the set of nodes in a given layer l and let $A_l(x) = \{\sigma \in N_l | \sigma(x) > 0\}$, where σ is the activation function of the underlying DF/DN. That is, A_l takes in an inference example as input and outputs the

Figure 3.1: Gaussian XOR samples used as a benchmark. Mixture of four Gaussians belonging to two classes. Class 0 consists of negative samples drawn from two Gaussians with means $(-1, -1)$ and $(1, 1)$, whereas class 1 comprises positive samples drawn from the other Gaussians with means $(1, -1)$ and $(-1, 1)$. Bayes optimal misclassification rate for this simulation setting is 0.28.

set of nodes *in a given layer* on which the input inference example activates. Now let $A_L = \bigoplus_{l=0}^{L} A_l$. That is, A takes in an inference example as input and outputs the concatenated set of nodes *in all layers up to and including layer* L on which the input inference example activates. Figure 3.2 is colored such that all points of the same color output the same value of A_L. These collections of points are partitions. More formally, the partition (i.e. cell) of a given inference point, x, is $C_L(x) = \{z \in \mathcal{X} | A_L(z) = A_L(x)\}$, where $\mathcal{X}$ is the domain of the DF/DN. In the case of DF–with L equal to the depth of the DF–this partition is simply a leaf cell. In the case of DN–with L equal to the total number of layers in the network—this partition is a convex polytope. Note that the partitions of input space defined by C_L are all disjoint, and that DN learns axis-oblique partitions in this example, even though they are not necessary.

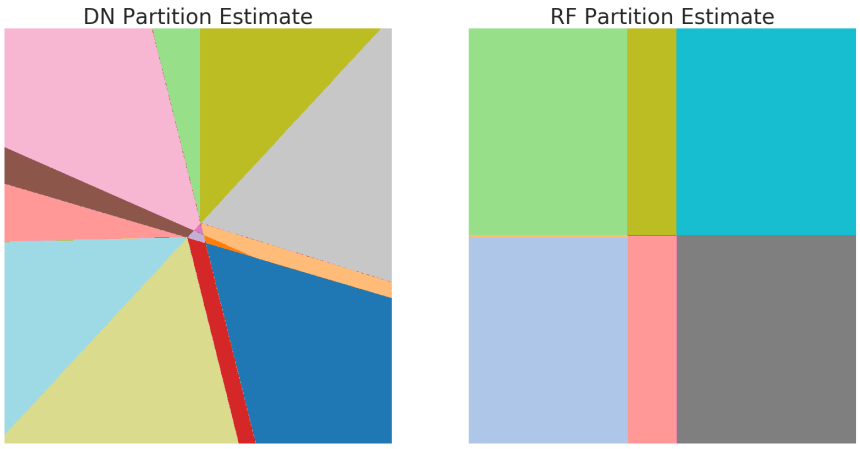

Figure 3.2: Visualizations of the partitions defined by C_L with respect to input space, $\mathcal{X}$, defined by a DN (left) and a DF (right). A unique region (i.e. partition) is visualized by an arbitrary unique color and corresponds to a region for which all values of C_L are equal. Note that the color value does not have any particular meaning.

In Figure 3.3, we visualize the layer-wise composition of C_L, the boundaries of C_l for all layers and for each node in each layer, and the effect of C_l on posteriors.

3.2 Internal representations of DN and DF

One way to explicitly understand P&D methods is through the fact that partition the feature space into parts. For example, consider decision trees, which explicitly partition feature space by making axis-aligned splits until the part (leaf nodes) are suitably pure or a stopping criterion is reached. To make a prediction for any sample given a specific tree, all that matters is the leaf node in which it lies, and the posterior function learned by the tree on that leaf node. Thus, the internal representation of a sample from the perspective of a decision tree is simply the indices of its leaves, or equivalently a one-hot encoding vector for the tree.

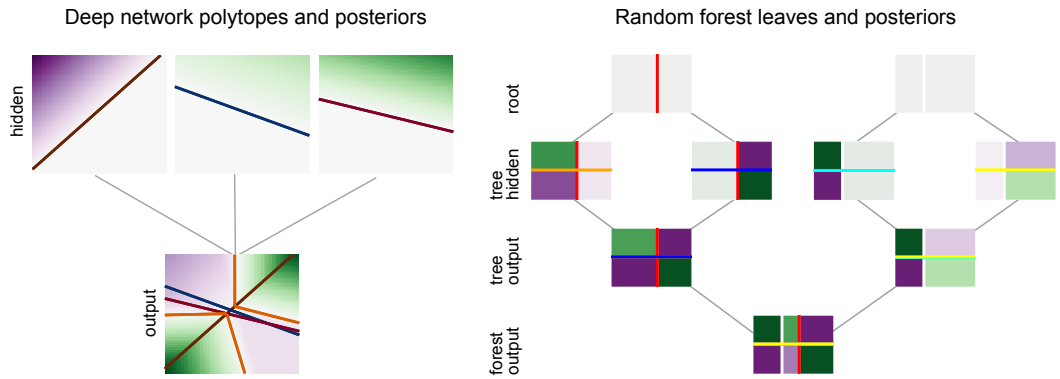

Figure 3.3: Visualizations of the Polytope Compositions for DF/DN. For the first layer/depth polytopes/leaves, we show the boundary for which C_0 changes value. That is, on one side of the visualized boundary the node is activated; on the other side, the node is not activated. In the case of DN, this means that one side contains the linear portion of the ReLU activation, while the other side contains the 0-valued region of the ReLU activation; in the case of DF, this indicates to which subsequent node (either left or right) examples will recursively fall. For all polytopes/leaves in all subsequent layers/depths, we visualize the C_l boundaries for that layer, and we also overlay the boundaries of $C_{l'}$ for all previous layers $l' < l$. The bottom rows both indicate the final model cells C_L. *Left*: In the base polytopes, the magnitude of the background color is determined by the activations of that layer, and the color is determined by the sign of the weight connecting that node to the final node (green indicates positive weight, purple indicates negative weight). In the output polytope, we color according to the sum of the activations of the previous layer, weighted according to the weights of the network's last layer. Note that the network posteriors are simply estimated by sending this sum of activations through a sigmoid function. *Right*: we color the background of all cells in all DF figs according to the empirical proportion of points within each that cell belonging to class 1 (green is 1, purple is 0).

Similarly, in deep network with ReLU activation functions, the subset of feature space uniquely activating a subset of activation functions forms a convex polytope (Montúfar et al., 2014; Arora et al., 2017; Hanin and Rolnick, 2019); within each of these polytopes, the output of the network is linear in the inputs. The internal representation of an arbitrary deep network can be defined from this perspectives. We define a region of a network as a learned convex polytope partition resulting from ReLU activations through the penultimate layer. Call this region *activated* if it contains at least one training sample and let L denote the number of activated regions. Any sample can be represented as a L-dimensional, one-hot encoding vector telling us which region it lies in. The value of the nonzero encoding element can be set to the true class label, predicted class label, or predicted posterior class probability. Given n samples, we can construct the matrix $Z \in \mathbb{R}^{n \times L}$. We then define the capacity of the network as $rank(Z) < min(n, L)$ and the effective dimesionality to be $trace(Z^T Z)$. Others have attempted to quantify the effective dimension, but of the parameters and not the internal representation. We propose using a norm-based control of the internal representation matrix Z.

In what follows, we first recall our problem setup using ReLU networks and review some relevant notions of complexity measures. We then explain our internal representation, which is generally applicable for `P&D` approaches. We demonstrate from comprehensive experiments that the matrix norm of our internal representation better predicts the generalization error than other existing measures.

3.2.1 Quantifying complexity in ReLU deep nets

Recall a d-layer feed-forward network with parameters w and Rectified Linear Unit (ReLU) as the activation function,

$$f_w(X) \triangleq \sigma\left(\ldots\left(\sigma\left(XW_1 + b_1\right)W_2 + b_2\right)\cdots\right)W_d + b_d,$$

where f_w is the function computed by the network, X is the training data, and $\sigma(z) = \max\{0, z\}$. For given x, let D_i denote the binary vector, with dimensionality equal to the number of nodes in layer i, with non-zeros indicating which nodes were activated by that x. Note that $D_i := D_i(X, w)$ depends on input X and weights w (from the previous $i-1$ layers).

In a fully-connected ReLU network with width k, there are k^d exponentially number of paths, i.e. the 'neural circuits' that correspond to different activation regions. However, only a subset of paths are *activated* from the training data, with D_i having nonzero entries. Thus, we want a data-dependent complexity measure, instead of capacity control. Moreover, we want this measure to generalize to P&D methods without weights w. We claim that the activation pattern of the data is all you need, which is represented via the internal representation matrix.

Definition 1 (Internal representation in ReLU neural network)**.** *For a network with activation matrices $\{D_i\}_{i=1}^d$, let $\mathcal{D} = [D_1, \cdots, D_d] \in R^{n\times(k_1+\cdots+k_d)}$ be the full (concatenated) activation matrix, where k_i denote the width of layer i. Then the total number of unique activated regions (polytopes) in $f_w(X)$ is given by the number of unique rows in $\mathcal{D}$, denoted as L. Now, construct the internal representation matrix $Z \in R^{n\times L}$, where the rows represents the data points, and the columns are the unique*

activated regions. $Z_{ij} = 1c_{ij}$ *if data point* i *is in the activated region* j*, and* 0 *elsewhere, where the value* c_{ij} *can be simply set as* 1*, chosen from true labels, predicted labels or predicted posterior probabilities.*

Claim 1. *For ReLU networks, internal representation* Z *serves as an activated path rank control without explicitly involving the weights* w *(since* D_i *depends on* w*).*

Proof. $L = \text{rank}(\mathcal{D})$ □

Claim 2. *Internal representation* Z *is a data-adaptive partition.*

Proof. Directly from Definition 1, Z partitions the data X as each data point (row of Z) is assigned to a unique activated region (column of Z). The partition assignment is learned by minimizing the empirical classification error of (X, Y, f_w). □

Alternatively, one can define the internal representation in ReLU network based on the penultimate layer activations as the following:

Definition 2. *(penultimate representation in ReLU neural network) For a network with activation matrices* $\{D_i\}_{i=1}^{d}$*, consider the last activation matrix* $D_d \in R^{n \times k_d}$*, where the bit-string with length* k_d *encodes the activation region of each data point. Then the total number of unique activated regions in* $f_w(X)$ *is given by the unique rows in* D_d*, denoted as* L*. The penultimate representation* $Z \in R^{n \times L}$ *can be constructed similarly as the internal representation matrix in Definition 1.*

Claim 3. *Let* H *be the number of unique affine functions in ReLU network* f_w*. Let* R *be the number of unique rows in* $D_d \in R^{n \times k_d}$*, and* N *be the unique rows in* $\mathcal{D} = [D_1, \cdots, D_d] \in R^{n \times (k_1 + \cdots + k_d)}$*. Then* $R \leq H \leq N$*.*

To analyze the model complexity, we can use matrix-norms of the internal representation Z as a complexity measure.

3.2.2 An empirical investigation of the P&D bounds

We numerically illustrate the value of the P&D framework using Deep Networks (DN) with ReLU as the activation function and Binary Cross Entropy as the loss function (Figure 3.4 (a)). Note that a single hidden layer with 3 nodes is sufficient to achieve Bayes optimal accuracy in this simulation. However, because the posterior are smooth, and ReLU deep nets approximate posteriors with piecewise affine functions, they cannot achieve zero cross-entropy loss on the test data. We consider two different architectures with increasing parameters: increasing width and increasing depth.

For the increasing width experiment, the architecture starts with 3 hidden layers of a single node each, and increases the number of nodes in the three hidden layers (left panels). When sweeping across widths—increasing the number of parameters by making each layer wider—the training error monotonically decreases as expected, while the testing error follows the classic U-shaped curve, also as expected.

For the increasing depth experiment, the architecture starts with 1 hidden layer consisting a single node and keeps growing until it consists 20 nodes, and it grows the depth of the network by adding layers, each with 20 nodes. When sweeping across depths—increasing the number of parameters by adding layers—the training error is no longer monotonically decreasing. Rather, training error decreases until the total number of parameters is larger than the

training data sample size, and then the training error *increases*. This suggests that increasing depth such that $p > n$ is implicitly regularizing, as even the training error increases. Moreover, in the increasing depth scenario, the test error follows the now famous double descent (a phenomenon where test error first gets better, then worse, and then gets better again), with training error peaking when the number of parameters is greater than sample size (at the same place that the training error changes its slope from negative to positive). Test error then rapidly decreases to near Bayes optimal.

These experiments collectively show that depth and width have different effects on the generalizability of a DN, even though they have the same number of parameters (see Figure 3.5). Specifically, although increasing width increases the capacity of the model, it overfits because it fails to reasonably control the complexity. On the other hand, increasing depth increases the capacity, but does so in such a way that complexity is controlled so that it achieve better generalization error.

Similarly, we use the Random Forest (RF) algorithm with Gini impurity as the split criterion (Figure 3.4 (b)). In the top row, the figure shows classification error $L(g_n)$ versus the total number of leaf nodes. In the RF algorithm with deep trees (left panels), a single decision tree is grown until each leaf node is pure, and then more trees are added with similar complexity as the first tree. These deep RFs exhibit the double-descent phenomenon, just like deep learning methods: as the number of leaf nodes increases, first test error drops, then it increases as it overfits, and then it drops again as more trees are included. In contrast, consider the RF algorithm with shallow trees (right

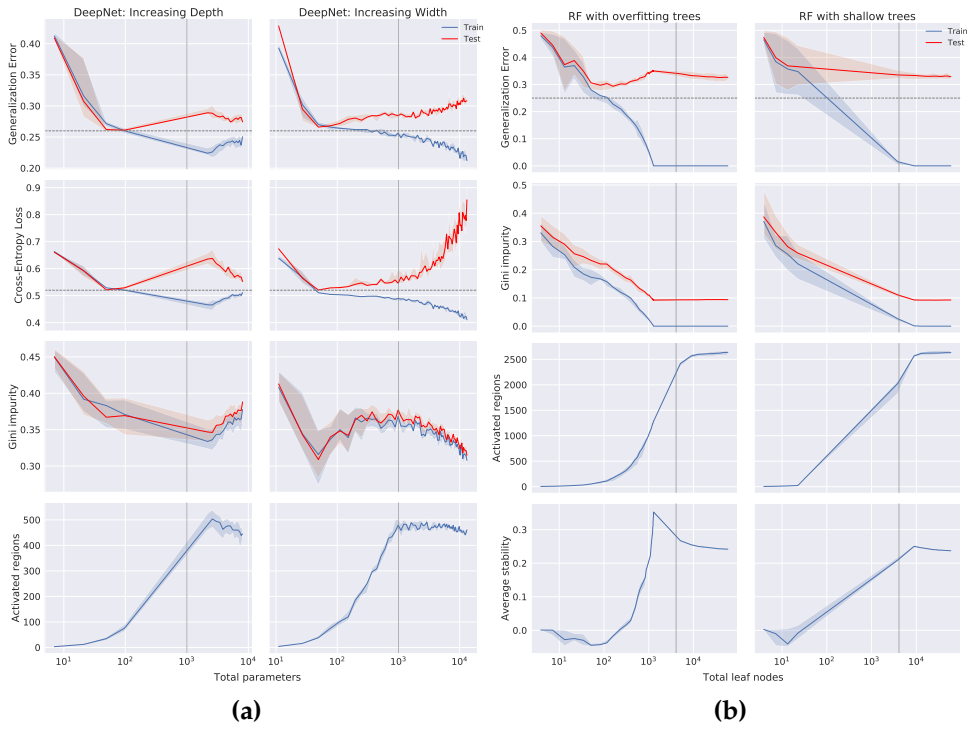

Figure 3.4: (a) Performance of DN with two different learning settings. Left: DeepNet: Increasing Width – a DN that starts with three hidden layers of a single node each and is grown iteratively increases the number of nodes in each layers. Right: DeepNet: Increasing Depth – a DN with a single hidden layer consisting 20 nodes, and grows the depth with the same number of nodes. The x-axis is the total number of parameters in the DN; the y-axis for the four rows are Cross-Entropy loss, Gini impurity, activation regions, and average stability, respectively. The dashed gray horizontal line in the first two rows indicates the optimal Bayes error, and optimal Cross Entropy loss, respectively. The solid gray vertical line indicates the number of data points. These results illustrate that the double descent phenomena only arises when using certain algorithms for growing the architecture, and plotting certain loss functions. **(b) Performance of RF with two different learning settings.** Left: Deep Tree RF – a single decision tree is grown iteratively until each leaf node is pure, and then more trees are added with similar complexity. Right: Shallow Tree RF – single decision tree is grown iteratively until depth is five, and then more trees are added with similar complexity. The x-axis is the total number of leaf nodes in the RF; the y-axis for the seven rows are classification error $L(g_n)$, Gini impurity, activation regions, and average stability, respectively. The dashed gray horizontal line in the first row indicates the optimal Bayes error. The solid gray vertical line indicates the number of data points. These results illustrate that the double descent phenomena only arises when using certain algorithms for growing the architecture, and plotting certain loss functions.

panels), that starts with a single tree increasing the number of its leaf nodes until it achieves a maximum depth of five, and then increases the number of trees and the maximum depth of each tree steadily. These shallow RFs exhibit the classical monotonically decreasing out-of-sample error rates. Notably, both algorithms demonstrate monotonically decreasing training error, and eventually both achieve Bayes optimal performance on the test data.

These results suggest that the reason behind the double descent phenomena is not the increasing complexity of the model as it is sometimes claimed (Nakkiran et al., 2020; Belkin et al., 2019), but rather the details of the growth model. Specifically, both the Deep Tree RF and Shallow Tree RF have the same "complexity" as defined by the total number of nodes (and therefore, parameters), but exhibit different generalization error curves. The same applies to the the DN Increasing Depth and DN Increasing Width. Hence, this indicates that a confounding variable causing the double descent (or not) is the algorithmic details for increasing the number of parameters, rather than some mysterious property of deep nets.

Moreover, we can see from the second row panels of the figure that the Gini impurity for both training and test set decreases across the model complexity in both settings. This is notable because in both settings, the forest greedily optimizes Gini impurity, rather than generalization error. It is well established that optimizing with respect to two similar metrics can yield dramatically different outcomes (Owhadi, Scovel, and Sullivan, 2015). Thus, perhaps it should not be surprising that Gini impurity and generalization error differ so much. Nonetheless, it suggests another complementary explanation for

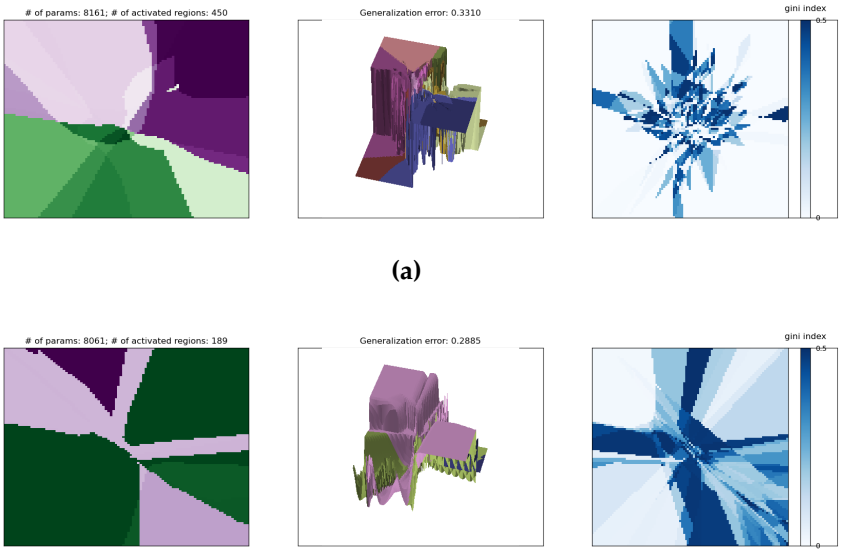

Figure 3.5: Effect of depth vs width on the generalizability of a DN models with the same number of parameters. (a) A model from the Increasing Width experiment. (b) A model from the Increasing Depth experiment. *Left*: Posterior polytope map. *Center*: Posterior surface plot. *Right*: Heatmap of the Gini impurity.

the double-descent curves: the loss that algorithms are implicitly optimizing when adding additional parameters may not be generalization error, but rather something else which is, in fact, monotonically decreasing.

3.2.3 Generalization in Deep Networks

Although modern machine learning have seen huge empirical success across a wide variety of tasks, why these models generalize well to unobserved data is still a mystery; and yet generalization is arguably the most fundamental aspect of machine learning (Neyshabur, Tomioka, and Srebro, 2015a; Recht et al., 2019).

Deep networks, even with over-parameterization, exhibit good generalization behavior. In an over-parametrized setting, the objective function has multiple global minima that minimize the training loss, but many of them might not generalize well. Therefore, just minimizing the training loss is not sufficient for learning since selecting the wrong global minima can lead to bad generalization behavior. In such situations, generalization behavior depends implicitly on the algorithm used to solve the objective function.

Generalization of deep networks has been of great interest in recent years, resulting in a number of theoretically and empirically motivated complexity measures that tries to explain the generalization ability of the DN models. As mentioned before, simply counting the number of parameters as a measure is not sufficient to predict and explain generalization. For linear models, norms and margin-based measures are commonly used for capacity control (Bartlett and Mendelson, 2001; Evgeniou, Pontil, and Poggio, 2000; Smola, Schölkopf, and Müller, 1998). Also norms such as the trace norm and max norm are considered as sensible inductive biases in matrix factorization and are often more appropriate than parameter-counting measures (Srebro and Shraibman, 2005; Srebro, Rennie, and Jaakkola, 2004). Bartlett and Mendelson (2001), Neyshabur, Tomioka, and Srebro (2015b), and Dziugaite and Roy (2017) suggested different (norm & margin)-based measures of network parameters to measure the capacity of DN. In a different line of work, Keskar et al., 2017 suggested "sharpness", i.e. robustness of the training error to perturbations in the parameters, as a complexity measure. Others, including McAllester (1999) and Dziugaite and Roy (2017) propose a PAC-Bayes analysis.

Jiang et al., 2020 explored a large scale study of generalization in deep networks and evaluated numerous measures that belong to different families: VC-dimension (Vapnik and Chervonenkis, 2013); norm- and margin- based bounds (Neyshabur, Tomioka, and Srebro, 2015b; Bartlett, Foster, and Telgarsky, 2017; Neyshabur et al., 2018); sharpness measures (Keskar et al., 2017); PAC-Bayes (McAllester, 1999; Dziugaite and Roy, 2017; Neyshabur et al., 2017); and path norm (Neyshabur, Salakhutdinov, and Srebro, 2015). From the extensive experiments, they concluded that sharpness-based measures such as PAC-Bayesian bounds outperform all the other measures.

As already discussed, we can use matrix norms of the internal representation matrix $Z \in R^{n \times L}$ as a complexity measure. The first matrix norm we used is the Ky Fan raw norm, which is the sum of singular values of the matrix Z, defined as,

$$\|Z\|_* = \sum_{i}^{\min\{n,L\}} |s_i(ZZ^T)|, \tag{3.1}$$

where s_i are the singular values. Thus, the Ky Fan $k - norm$ is defined as the sum of the k largest singular values. Ky Fan kernel norm is same as the raw norm but with the matrix ZZ^T. The second matrix norm we used is the Schatten norm. Given a real number $p \geq 1$, the Schatten $p - norm$ of the internal representation matrix Z is defined as the vector $p - norm$ of the vector of the singular values of Z:

$$\|Z\|_{S_p} = \left(\sum_{i=1} s_i(Z)^p \right)^{\frac{1}{p}}. \tag{3.2}$$

Finally, we can use a generalization of both norms, i.e. the vector $p - norm$ of the first k singular values. We call this Ky Fan-Schatten norm and is defined

as,

$$\|Z\|_{(p,k)} = \left(\sum_{i=1}^{k} s_i(Z)^p \right)^{\frac{1}{p}}. \tag{3.3}$$

Thus, we evaluate our complexity measures against these state-of-the-art complexity measures.

3.2.3.1 Evaluation criteria

To evaluate the correlation of a complexity measure to a generalization gap, i.e the difference between the test error and training error, we will use the Kendall's Rank-Correlation Coefficient (Kendall, 1938), as in Jiang et al., 2020. Given a set of trained models with hyper-parameters in the set Θ, their associated values of the complexity measure $\{c(\theta) | \theta \in \Theta\}$, and their respective generalization gap $\{g(\theta) | \theta \in \Theta\}$, the goal is to evaluate how consistent a measure is with the empirically observed generalization of the models. Let $\mathcal{M}$ be a set of the trained models. Each element has the form of a pair: complexity measure c versus generalization gap g.

$$\mathcal{M} \triangleq \cup_{\theta \in \Theta} \{(c(\theta), g(\theta)\}$$

Therefore, we use Kendall's rank coefficient τ to capture to what degree such consistency holds among the elements of $\mathcal{M}$.

$$\tau(\mathcal{M}) \triangleq \frac{1}{|\mathcal{M}|(|\mathcal{M}|-1)} \sum_{(c_1,g_1) \in \mathcal{M}} \sum_{(c_2,g_2) \in \mathcal{M} \setminus (c_1,g_1)} \text{sign}(c_1 - c_2) \, \text{sign}(g_1 - g_2)$$

We tested the above mentioned complexity measures and our matrix norm of the internal representation on our Gaussian XOR experiments. Here, $\mathcal{M}$ denotes the set of DN models with increasing depth experiments as stated

before. We illustrate that the matrix norms of the internal representation correlate more strongly with generalization gap than the existing state-of-the-art complexity measures, such as PAC-Bayes based measures (see Table 3.2).

Table 3.1: Complexity measures, reference to the equation, and the rank-correlation coefficients with DN models (increasing depth) trained on the Gaussian XOR dataset.

Complexity Measures	**Reference**	**Kendal's τ**	**p value**
Parameters	Jiang et al., 2020, Eq. 20	-0.2624	0.0000
Product of Frobenius	Jiang et al., 2020, Eq. 37	-0.2504	0.0000
Product of Spectral	Jiang et al., 2020, Eq. 32	-0.2481	0.0000
Frobenius/Spectral	Jiang et al., 2020, Eq. 33	-0.2414	0.0000
Product of Frobenius/Margin	Jiang et al., 2020, Eq. 36	-0.2321	0.0000
Spectral INIT MAIN	Jiang et al., 2020, Eq. 29	-0.2243	0.0000
Spectral ORIG MAIN	Jiang et al., 2020, Eq. 30	-0.2241	0.0000
Product of Spectral/Margin	Jiang et al., 2020, Eq. 34	-0.2212	0.0000
Frobenius Distance	Jiang et al., 2020, Eq. 40	-0.1988	0.0000
PAC-Bayes Magnitude INIT	Jiang et al., 2020, Eq. 56	-0.1907	0.0001
Sum of Frobenius	Jiang et al., 2020, Eq. 39	-0.1885	0.0001
PAC-Bayes Magnitude ORIG	Jiang et al., 2020, Eq. 57	-0.1850	0.0002
Parameter Norm	Jiang et al., 2020, Eq. 42	-0.1807	0.0002
Path Norm	Jiang et al., 2020, Eq. 44	-0.1637	0.0002
Sum of Frobenius/Margin	Jiang et al., 2020, Eq. 36	-0.1309	0.0075
Distance Spectral INIT	Jiang et al., 2020, Eq. 41	-0.1129	0.0211
Sum of Spectral	Jiang et al., 2020, Eq. 35	-0.0854	0.0810
Path Norm/Margin	Jiang et al., 2020, Eq. 43	-0.0819	0.0946
Sum of Spectral/Margin	Jiang et al., 2020, Eq. 34	-0.0055	0.9101
PAC-Bayes Magnitude Flatness	Jiang et al., 2020, Eq. 61	0.0343	0.4951
PAC-Bayes Flatness	Jiang et al., 2020, Eq. 53	0.1512	0.0025
PAC-Bayes INIT	Jiang et al., 2020, Eq. 48	0.1635	0.0008
PAC-Bayes ORIG	Jiang et al., 2020, Eq. 49	0.1666	0.0007
Inverse Margin	Jiang et al., 2020, Eq. 22	0.4214	0.0000
[KF-kernel-3 [ours]]	Equation 3.1	[0.4770]	[0.0013]
[KF-kernel-5 [ours]]	Equation 3.1	[0.4777]	[0.0013]
[KF-kernel-4 [ours]]	Equation 3.1	[0.4792]	[0.0008]
[Schatten-4 [ours]]	Equation 3.2	[0.4865]	[0.0008]
[KF-Schatten-3 [ours]]	Equation 3.3	[0.4939]	[0.0010]

Table 3.2: Complexity measures, reference to the equation, and the rank-correlation coefficients with DN models (both increasing depth and increasing width) trained on the Gaussian XOR dataset.

Complexity Measures	Reference	Kendal's τ	p value
Frobenius/Spectral	Jiang et al., 2020, Eq. 33	-0.0235	0.8886
Parameters	Jiang et al., 2020, Eq. 20	-0.0117	0.9442
PAC-Bayes Magnitude ORIG	Jiang et al., 2020, Eq. 57	0.0352	0.8336
Product of Frobenius/Margin	Jiang et al., 2020, Eq. 36	0.0352	0.8336
PAC-Bayes Magnitude INIT	Jiang et al., 2020, Eq. 56	0.0352	0.8336
Product of Frobenius	Jiang et al., 2020, Eq. 37	0.0704	0.6744
Inverse Margin	Jiang et al., 2020, Eq. 22	0.1056	0.5286
PAC-Bayes Magnitude Flatness	Jiang et al., 2020, Eq. 61	0.1382	0.4177
Product of Spectral	Jiang et al., 2020, Eq. 32	0.1408	0.4008
Spectral ORIG MAIN	Jiang et al., 2020, Eq. 30	0.1408	0.4008
Product of Spectral/Margin	Jiang et al., 2020, Eq. 34	0.1525	0.3627
Spectral INIT MAIN	Jiang et al., 2020, Eq. 29	0.1525	0.3627
Parameter Norm	Jiang et al., 2020, Eq. 42	0.2698	0.1073
Sum of Frobenius/Margin	Jiang et al., 2020, Eq. 36	0.2815	0.0929
Sum of Frobenius	Jiang et al., 2020, Eq. 39	0.2815	0.0929
Distance Spectral INIT	Jiang et al., 2020, Eq. 41	0.2933	0.0801
Sum of Spectral	Jiang et al., 2020, Eq. 35	0.2933	0.0801
PAC-Bayes Flatness	Jiang et al., 2020, Eq. 53	0.2976	0.0790
PAC-Bayes ORIG	Jiang et al., 2020, Eq. 49	0.3050	0.0687
Sum of Spectral/Margin	Jiang et al., 2020, Eq. 34	0.3167	0.0587
PAC-Bayes INIT	Jiang et al., 2020, Eq. 48	0.3519	0.0357
Frobenius Distance	Jiang et al., 2020, Eq. 40	0.3754	0.0251
Path Norm	Jiang et al., 2020, Eq. 44	0.4927	0.0033
Path Norm/Margin	Jiang et al., 2020, Eq. 43	0.5044	0.0026
[KF-raw-1 [ours]]	Equation 3.1	[0.5428]	[0.0013]
[KF-kernel-1 [ours]]	Equation 3.1	[0.5428]	[0.0013]
[Schatten-9 [ours]]	Equation 3.2	[0.5439]	[0.0008]
[Schatten-10 [ours]]	Equation 3.2	[0.5439]	[0.0008]

Chapter 4

Discussion and Conclusion

Artificial Intelligence has made remarkable advances, achieving capabilities previously thought to be impossible just a few decades ago, largely by leveraging machine learning which provides a rich formalism for characterizing learning and intelligence. These advances have come with considerably increasing economic and ecological costs, but are recently yielding only incremental improvements. This motivates a research strategy that complements the mainstream view of accruing increasingly larger datasets and training increasingly larger models. Thus, in this thesis, we propose 'Partition and Decode' (P&D), a framework for understanding a wide variety of modern machine learning methods—including decision trees, random forests, gradient boosting trees, and deep nets with ReLU activation functions. The key idea is that these learning methods explicitly or implicitly partitions feature space into parts, and thus, data can be encoded by which part it resides in. This framework motivated the study of the internal representations of each of these approaches via considering the 'neural circuits' that correspond to different activation regions. Our experimental evaluations demonstrated that

certain matrix norms of this representation correlate more strongly with generalization gap than existing complexity measures, such as PAC-Bayes based measures and Path-Norm.

The P&D framework helps to understand and explain certain phenomena, such as the double descent paradox. The double descent risk curve, a phenomenon where test error first gets better, then worse, and then gets better again, was recently noticed by the deep learning community and has perplexed the machine learning world since its introduction a few years ago. Specifically, a recent PNAS article began with, "Deep learning methodology has revealed a surprising statistical phenomenon: overfitting can perform well" (Bartlett et al., 2020). The classic textbook, Elements of Statistical Learning (Friedman, Hastie, and Tibshirani, 2001) further argues, "interpolating fits…[are] unlikely to predict future data well at all". We contend that neither of these are true. That overfitting can perform well has been established in the statistics and machine learning literature already for decades. Specifically, there are many examples in the statistics and machine learning literature in which interpolating classifiers, that is, classifiers with zero training error, predict future data well. What has not be clear, however, is how two seemingly incompatible facts can both be true: first, that increasing complexity eventually leads to worse out-of-sample performance (bias/variance or bias/complexity trade-off); and second, that interpolating classifiers (which are incredibly complex and often have no asymptotic bias) perform extremely well.

The key intuition to resolving this apparent paradox is that not all parameters are created equally. More specifically, sometimes parameters only

increase representational capacity (thereby reducing bias), sometimes they only 'regularize' (thereby reducing complexity), and sometimes they do a bit of both. This fact is well-known to all machine learning practitioners. Consider an over-parameterized linear regression problem (where the number of dimensions is larger than the number of samples): what do we do? We add a parameter which penalizes fits that are not adequately smooth, thereby regularizing and reducing complexity. This parameter clearly does not add any additional representational capacity, so this example is an existence proof that certain parameters purely reduce complexity. While the dual role of parameters is clear in linear regression, the extent to which an additional parameter reduce bias or complexity is more complicated and confusing in nonlinear models such as random forests and deep nets. And while it is true that the regularization term is a hyper-parameter, it is also true that the architecture is a hyper-parameter.

Despite many empirical and theoretical investigations into its nature, we still lack a clear intuitive explanation that (i) fits our understanding of the bias-variance trade-off, and (ii) accounts for the observed paradoxical phenomenon. We argue that the key to resolving this internal conflict is to appreciate that parameters can add both capacity and regularization, once both terms are suitably defined. In simple linear regression scenarios, the extent to which each parameter serves which function is clear, for example, adding a penalty term adds a single parameter which purely regularizes, and therefore, does not change capacity at all. Yet in modern machine learning methods, such as random fourier features, random forests, and deep networks, the relationship

between number of samples, number of parameters, capacity, and regularization is often unclear and implicit. We argue that the effective dimensionality of the internal representation of the data is a suitable and principled method for calculating the capacity in these overparameterized algorithms. This capacity estimate accounts for both explicit and implicit regularization, including penalty terms, early stopping, approximate algorithms, and architecturally induced regularization. A key to the success of this explanation is realizing that deep nets, decision trees, and random forests all partition feature space. Therefore, a reasonable and lossless internal representation of a given input to these methods, regardless of their representational architecture or learning algorithm, is its one-hot sparsely encoded activation vector. We show that with this definition, capacity monotonically increases until the number parameters is approximately equal to the number of samples, and then it saturates because the internal representation is then full rank. At that point, if adding additional parameters decreases the variance of the internal representation—which penalty terms do—generalization error begins to decrease.

We have also shown that the double descent curve can be an artifact of the particular rules for adding parameters, quantifying complexity, and implicit loss optimization by adding more parameters. This is because additional trees in a random forest with deep trees are effectively regularizing, more than they are adding capacity, much like adding an L_2 penalty in linear regression. Thus, while often the number of parameters is a good proxy for the complexity or capacity of a decision rule, in deep nets, as in random forests and other penalized classification/regression settings, depending on

how one increases the number of parameters, those parameters can increase fit or regularization. Formalizing and proving the above is of interest for future work. Moreover, regardless of how one increases parameters, the loss function that DF is actually optimizing, Gini impurity, exhibits the expected monotonic behavior. Recall that in these double descent curves, there are in fact two procedures at play: one for adding nodes, and another for optimizing the weights of the existing nodes. It is interesting to note that to achieve these double descent curves in various other publications, the algorithms for growing the deep networks are often implicit. This suggests another avenue of investigation for explaining the double-descent in deep networks would be to better understand algorithms for growing networks, much like the theory already in place for growing trees (Ben-Haim and Tom-Tov, 2010; Hulten, Spencer, and Domingos, 2001; Beygelzimer, Kakade, and Langford, 2006).

Bibliography

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*, pp. 1097–1105.

Brown, Tom B, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). "Language Models are Few-Shot Learners". In: arXiv: 2005.14165 [cs.CL].

Zhang, Yu, James Qin, Daniel S Park, Wei Han, Chung-Cheng Chiu, Ruoming Pang, Quoc V Le, and Yonghui Wu (2020). "Pushing the Limits of Semi-Supervised Learning for Automatic Speech Recognition". In: arXiv: 2010.10504 [eess.AS].

Statt, Nick (2019). *OpenAI's Dota 2 AI steamrolls world champion e-sports team with back-to-back victories*. https://www.theverge.com/2019/4/13/18309459/openai-five-dota-2-finals-ai-bot-competition-og-e-sports-the-international-champion.

Caruana, Rich and Alexandru Niculescu-Mizil (2006). "An Empirical Comparison of Supervised Learning Algorithms". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: ACM, pp. 161–168.

Caruana, Rich, Nikos Karampatziakis, and Ainur Yessenalina (2008). "An empirical evaluation of supervised learning in high dimensions". In: *Proceedings of the 25th international conference on Machine learning*. New York, New York, USA: ACM, pp. 96–103.

Fernández-Delgado, Manuel, Eva Cernadas, Senén Barro, and Dinani Amorim (2014). "Do we need hundreds of classifiers to solve real world classification problems". In: *J. Mach. Learn. Res.* 15.1, pp. 3133–3181.

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, pp. 785–794.

Patel, Ankit B, Tan Nguyen, and Richard G Baraniuk (2015). "A Probabilistic Theory of Deep Learning". In: arXiv: 1504.00641 [stat.ML].

Zhou, Zhi-Hua and Ji Feng (2018). "Deep forest". en. In: *Natl Sci Rev* 6.1, pp. 74–86.

Shen, Wei, Yilu Guo, Yan Wang, Kai Zhao, Bo Wang, and Alan Yuille (2019). "Deep Differentiable Random Forests for Age Estimation". In: arXiv: 1907.10665 [cs.CV].

Carreira-Perpinan, Miguel A and Pooya Tavallali (2018). "Alternating optimization of decision trees, with application to learning sparse oblique trees". In: *Advances in Neural Information Processing Systems 31*. Ed. by S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett. Curran Associates, Inc., pp. 1218–1228.

Hehn, Thomas M and Fred A Hamprecht (2019). "End-to-End Learning of Deterministic Decision Trees". In: *Pattern Recognition*. Springer International Publishing, pp. 612–627.

Priebe, Carey E., Joshua T. Vogelstein, Florian Engert, and Christopher M. White (2020). "Modern Machine Learning: Partition & Vote". In: *bioRxiv*. DOI: 10.1101/2020.04.29.068460. eprint: https://www.biorxiv.org/content/early/2020/09/19/2020.04.29.068460.full.pdf. URL: https://www.biorxiv.org/content/early/2020/09/19/2020.04.29.068460.

Stone, C. (1977). "Consistent nonparametric regression". In: *Annals of Statistics* 5.2, pp. 595–645.

Devroye, L., L. Györfi, and G. Lugosi (1997). *A Probabilistic Theory of Pattern Recognition*. Springer.

Breiman, Leo (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

Criminisi, A. and J. Shotton (2013). *Decision Forests for Computer Vision and Medical Image Analysis*. Springer. ISBN: 1447149289.

Athey, Susan, Julie Tibshirani, and Stefan Wager (2019). "Generalized random forests". In: *Annals of Statistics* 47.2, pp. 1148–1178. DOI: 10.1214/18-AOS1709. URL: https://doi.org/10.1214/18-AOS1709.

Tomita, Tyler M., James Browne, Cencheng Shen, Jaewon Chung, Jesse L. Patsolic, Benjamin Falk, Jason Yim, Carey E. Priebe, Randal Burns, Mauro Maggioni, and Joshua T. Vogelstein (2020). "Sparse Projection Oblique Randomer Forests". In: *Journal of Machine Learning Research* to appear. arXiv: 1506.03410.

Hastie, T., R. Tibshirani, and J. Friedman (2001). "The Elements of Statistical Learning". In:

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep Learning". In: *Nature* 521.7553, pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

Sze, Vivienne, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer (2017). "Efficient processing of deep neural networks: A tutorial and survey". In: *Proceedings of the IEEE* 105.12, pp. 2295–2329. URL: https://ieeexplore.ieee.org/abstract/document/8114708.

Montúfar, Guido, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio (2014). "On the Number of Linear Regions of Deep Neural Networks". In: *Proceedings of the International Conference on Neural Information Processing Systems*, 2924—2932.

Montúfar, Guido (2017). "Notes on the number of linear regions of deep neural networks". In: *Sampling Theory and Applications*. Tallinn, Estonia.

Hornik, K., M. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2, pp. 359–366.

Zhang, Chiyuan, S. Bengio, Moritz Hardt, B. Recht, and Oriol Vinyals (2017). "Understanding deep learning requires rethinking generalization". In: *ArXiv* abs/1611.03530.

Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal (2019). "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116.32, pp. 15849–15854. ISSN: 0027-8424. DOI: 10.1073/pnas.1903070116. eprint: https://www.pnas.org/content/116/32/15849.full.pdf. URL: https://www.pnas.org/content/116/32/15849.

Belkin, Mikhail, Daniel J Hsu, and Partha Mitra (2018). "Overfitting or perfect fitting? Risk bounds for classification and regression rules that interpolate". In: *Advances in Neural Information Processing Systems*. Ed. by S Bengio, H

Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett. Vol. 31. Curran Associates, Inc.

Bartlett, Peter L, Philip M Long, Gábor Lugosi, and Alexander Tsigler (2020). "Benign overfitting in linear regression". en. In: *Proceedings of the National Academy of Sciences* 117, pp. 30063 –30070.

Arora, R., A. Basu, Poorya Mianjy, and A. Mukherjee (2017). "Understanding Deep Neural Networks with Rectified Linear Units". In: *ArXiv* abs/1611.01491.

Hanin, B. and D. Rolnick (2019). "Complexity of Linear Regions in Deep Networks". In: *ArXiv* abs/1901.09021.

Nakkiran, Preetum, Gal Kaplun, Yamini Bansal, Tristan Yang, B. Barak, and Ilya Sutskever (2020). "Deep Double Descent: Where Bigger Models and More Data Hurt". In: *ArXiv* abs/1912.02292.

Owhadi, Houman, Clint Scovel, and Tim Sullivan (2015). "On the Brittleness of Bayesian Inference". In: *SIAM Review* 57.4, pp. 566–582.

Neyshabur, Behnam, Ryota Tomioka, and Nathan Srebro (2015a). "In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning". In: *CoRR* abs/1412.6614.

Recht, B., R. Roelofs, Ludwig Schmidt, and Vaishaal Shankar (2019). "Do ImageNet Classifiers Generalize to ImageNet?" In: *ArXiv* abs/1902.10811.

Bartlett, P. and S. Mendelson (2001). "Rademacher and Gaussian Complexities: Risk Bounds and Structural Results". In: *J. Mach. Learn. Res.*

Evgeniou, T., M. Pontil, and T. Poggio (2000). "Regularization Networks and Support Vector Machines". In: *Advances in Computational Mathematics* 13, pp. 1–50.

Smola, Alex, B. Schölkopf, and K. Müller (1998). "The connection between regularization operators and support vector kernels". In: *Neural networks : the official journal of the International Neural Network Society* 11 4, pp. 637–649.

Srebro, Nathan and Adi Shraibman (2005). "Rank, Trace-Norm and Max-Norm". In: *COLT*.

Srebro, Nathan, Jason D. M. Rennie, and T. Jaakkola (2004). "Maximum-Margin Matrix Factorization". In: *NIPS*.

Neyshabur, Behnam, Ryota Tomioka, and Nathan Srebro (2015b). "Norm-Based Capacity Control in Neural Networks". In: *COLT*.

Dziugaite, G. and Daniel M. Roy (2017). "Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data". In: *ArXiv* abs/1703.11008.

Keskar, N., D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang (2017). "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *ArXiv* abs/1609.04836.

McAllester, David A. (1999). "PAC-Bayesian model averaging". In: *COLT '99*.

Jiang, Yiding, Behnam Neyshabur, H. Mobahi, Dilip Krishnan, and S. Bengio (2020). "Fantastic Generalization Measures and Where to Find Them". In: *ArXiv* abs/1912.02178.

Vapnik, V. and A. Chervonenkis (2013). "On the Uniform Convergence of the Frequencies of Occurrence of Events to Their Probabilities". In: *Empirical Inference*.

Bartlett, P., Dylan J. Foster, and Matus Telgarsky (2017). "Spectrally-normalized margin bounds for neural networks". In: *NIPS*.

Neyshabur, Behnam, Srinadh Bhojanapalli, David A. McAllester, and Nathan Srebro (2018). "A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks". In: *ArXiv* abs/1707.09564.

Neyshabur, Behnam, Srinadh Bhojanapalli, D. McAllester, and Nathan Srebro (2017). "Exploring Generalization in Deep Learning". In: *NIPS*.

Neyshabur, Behnam, R. Salakhutdinov, and Nathan Srebro (2015). "Path-SGD: Path-Normalized Optimization in Deep Neural Networks". In: *NIPS*.

Kendall, M. (1938). "A NEW MEASURE OF RANK CORRELATION". In: *Biometrika* 30, pp. 81–93.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2001). *The elements of statistical learning*. Vol. 1. Springer Series in Statistics.

Ben-Haim, Yael and Elad Tom-Tov (2010). "A Streaming Parallel Decision Tree Algorithm". In: *Journal of Machine Learning Research* 11.Feb, pp. 849–872.

Hulten, Geoff, Laurie Spencer, and Pedro Domingos (2001). "Mining time-changing data streams". In: *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '01. San Francisco, California: Association for Computing Machinery, pp. 97–106.

Beygelzimer, Alina, Sham Kakade, and John C Langford (2006). "Cover trees for nearest neighbor". In: *Proceedings of the International Conference on Machine Learning*. Vol. 1. ACM Press, pp. 97–104.